SIAM J. MATRIX ANAL. APPL.
Vol. 28, No. 2, pp. 477–502

FAST METHODS FOR ESTIMATING THE DISTANCE TO UNCONTROLLABILITY*

M. GU†, E. MENGI‡, M. L. OVERTON‡, J. XIA†, AND J. ZHU†

Abstract. The distance to uncontrollability for a linear control system is the distance (in the 2-norm) to the nearest uncontrollable system. We present an algorithm based on methods of Gu and Burke–Lewis–Overton that estimates the distance to uncontrollability to any prescribed accuracy. The new method requires $O(n^4)$ operations on average, which is an improvement over previous methods which have complexity $O(n^6)$, where n is the order of the system. Numerical experiments indicate that the new method is reliable in practice.

Key words. distance to uncontrollability, complex controllability radius, trisection, real eigenvalue extraction, shifted inverse iteration, shift-and-invert Arnoldi, Sylvester equation, Kronecker product

AMS subject classifications. 65F15, 93B05, 65K10

DOI. 10.1137/05063060X

1. Introduction.

Given $A \in \mathbb{C}^{n\times n}, B \in \mathbb{C}^{n\times m}$, the linear control system

$$\dot{x} = Ax + Bu \tag{1.1}$$

is controllable if for every pair of states $x_0, x_f \in \mathbb{C}^n$ there exists a continuous control function $u(t)$ to steer the initial state x_0 to the final state x_f within finite time. Equivalently, according to a well-known result by Kalman [8], the system (1.1) is controllable if the matrix $[A - \lambda I \;\; B]$ has full row rank for all $\lambda \in \mathbb{C}$.

To measure the conditioning of (1.1), the distance to uncontrollability was introduced in [12] as

$$\tau(A, B) = \min\{\|\Delta A \; \Delta B\| \; : \; (A + \Delta A, B + \Delta B) \text{ is uncontrollable}\}, \tag{1.2}$$

which was later shown to be equivalent to [4, 5]

$$\tau(A, B) = \min_{\lambda\in\mathbb{C}} \sigma_n([A - \lambda I \;\; B]), \tag{1.3}$$

where $\|\cdot\|$ denotes the 2-norm or Frobenius norm[1] and $\sigma_n([A - \lambda I \;\; B])$ denotes the nth largest singular value of the $n \times (n + m)$ matrix $[A - \lambda I \;\; B]$. This is a global nonsmooth optimization problem in two real variables α and β, the real and imaginary parts of λ. But note that $\sigma_n([A - \lambda I \;\; B])$ is not convex and may have many local minima, so standard optimization methods, which are guaranteed only to converge to a local minimum, will not yield reliable results in general.

*Received by the editors May 3, 2005; accepted for publication (in revised form) by N. Mastronardi December 21, 2005; published electronically June 21, 2006.
http://www.siam.org/journals/simax/28-2/63060.html

†Department of Mathematics, University of California at Berkeley, Berkeley, CA 94720 (mgu@math.berkeley.edu, jxia@math.berkeley.edu, zhujiang@math.berkeley.edu). The research of these authors was supported in part by the National Science Foundation grant DMS-0412049.

‡Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 (mengi@cs.nyu.edu, overton@cs.nyu.edu). The research of these authors was supported in part by the National Science Foundation grant CCR-0204388.

[1]The definitions of $\tau(A, B)$ in terms of the Frobenius norm and the 2-norm are equivalent.

Gu [6] proposed a bisection method which can correctly estimate $\tau(A, B)$ within a factor of 2 in time polynomial in n. Throughout the paper, when we refer to operation counts we assume that the computation of the eigenvalues of a matrix or pencil is an atomic operation whose cost is cubic in the dimension. Burke, Lewis, and Overton [3] suggested a trisection variant to retrieve the distance to uncontrollability to any desired accuracy. The methods in these two papers are based on a simultaneous comparison of two estimates $\delta_1 > \delta_2$ with $\tau(A, B)$. More precisely, Gu derived a scheme that returns one of the inequalities

$$\tau(A, B) \leq \delta_1 \tag{1.4}$$

and

$$\tau(A, B) > \delta_2. \tag{1.5}$$

Even if both of the inequalities are satisfied, Gu's scheme returns information about only one of the inequalities. Gu's method depends on the extraction of the real eigenvalues of a pencil of size $2n^2 \times 2n^2$ and the imaginary eigenvalues of matrices of size $2n \times 2n$. Computationally the verification scheme is dominated by the extraction of the real eigenvalues of the generalized problem of size $2n^2 \times 2n^2$, which requires $O(n^6)$ operations if the standard QZ algorithm is used.

In this paper we present an alternative verification scheme for comparisons (1.4) and (1.5). In the new verification scheme we still need to find real eigenvalues of $2n^2 \times 2n^2$ matrices, so there is no asymptotic gain over Gu's verification scheme when we use the QR algorithm. Nevertheless, we show that the inverse of these $2n^2 \times 2n^2$ matrices shifted by a real number times the identity can be multiplied onto a vector efficiently by solving a Sylvester equation of size $2n$ with a cost of $O(n^3)$. Therefore, given a real number as the shift, by applying shifted inverse iteration or a shift-and-invert preconditioned Arnoldi method, the closest eigenvalue to the real number can be obtained by performing $O(n^3)$ operations. Motivated by the fact that we need only real eigenvalues, we provide two alternative ways to scan the real axis to find the desired eigenvalues. Both of the approaches require an upper bound on the norm of the input matrix (of size $2n^2 \times 2n^2$) as a parameter. For one of the approaches, which is based on a "divide and conquer" idea, choosing this parameter arbitrarily large does not affect the efficiency of the algorithm much. The efficiency of the other approach, which we name "adaptive progress," depends not only on this parameter significantly but also on another parameter that bounds the distance between the closest pair of eigenvalues from below. For the divide and conquer approach, we prove that extracting all of the real eigenvalues requires $O(n^4)$ operations on average and $O(n^5)$ operations in the worst case. For the adaptive progress approach such neat results are not immediate because of the dependence of the performance of the algorithm on the parameters. In practice we observe that the divide and conquer approach is the more efficient and more reliable method.

In section 2 we will review the trisection method for estimating $\tau(A, B)$ and Gu's scheme for verifying which one of (1.4) and (1.5) holds. In section 3 we present our modified eigenvalue problem for the same purpose and fast methods based on the shifted inverse iteration or shift-and-invert Arnoldi for solving it. Specifically, to extract all of the real eigenvalues, we discuss two search strategies: an adaptive progress approach and a divide and conquer approach. The effectiveness and reliability of the methods are demonstrated by the numerical examples in section 4.

2. Trisection and Gu's verification scheme.

2.1. Bisection and trisection. The problem of computing the distance to uncontrollability is equivalent to the minimization of $\sigma_n([A - \lambda I \quad B])$ over the entire complex plane. Gu [6] proposed the first polynomial-time estimation scheme. Burke, Lewis, and Overton [3] later suggested a trisection version to retrieve the distance to uncontrollability to an arbitrary accuracy. Given two real numbers $\delta_1 > \delta_2$, at each iteration both of the algorithms alter an upper bound or a lower bound depending on which of the inequalities (1.4) and (1.5) holds. This test is based on the following theorem [6], which is a consequence of the fact that singular values are well-conditioned (in the absolute sense).

THEOREM 2.1 (see Gu [6]). *Assume that $\delta > \tau(A, B)$. Given an $\eta \in [0, 2(\delta - \tau(A, B))]$, there exist at least two pairs of real numbers α and β such that*

$$\delta \in \sigma\left([A - (\alpha + \beta i)I, B]\right) \text{ and } \delta \in \sigma\left([A - (\alpha + \eta + \beta i)I, B]\right), \tag{2.1}$$

where $\sigma(\cdot)$ denotes the set of singular values of its argument.

We shall describe two alternative ways of verifying the existence of a pair α and β satisfying (2.1) for a given δ and η in subsections 2.2 and 3.1. Suppose we set $\delta_1 = \delta$ and $\delta_2 = \delta - \eta/2$. The theorem above implies that when no pair satisfying (2.1) exists the inequality $\eta > 2(\delta - \tau(A, B))$ is satisfied, so condition (1.5) holds. On the other hand, when a pair exists, then by definition (1.3) we can conclude (1.4).

Gu's bisection algorithm (Algorithm 1) keeps only an upper bound on the distance to uncontrollability. It refines the upper bound until condition (1.5) is satisfied. Notice that in Algorithm 1, $\delta = \eta = \delta_1$. At termination the distance to uncontrollability lies within factor of 2 of δ_1, with $\delta_1/2 < \tau(A, B) \leq 2\delta_1$.

Algorithm 1 Gu's bisection estimation algorithm

Call:	$\delta_1 \leftarrow$ `Bisection`(A,B).
Input:	$A \in \mathbb{C}^{n\times n}$ and $B \in \mathbb{C}^{n\times m}$ with $m \leq n$.
Output:	A scalar δ_1 satisfying $\delta_1/2 < \tau(A, B) \leq 2\delta_1$.

1. Initialize the estimate as $\delta_1 \leftarrow \sigma_n([A \;\; B])/2$.
   repeat
     $\delta_2 \leftarrow \frac{\delta_1}{2}$.
     Apply Gu's test.
     if (1.4) is verified **then**
       $\delta_1 \leftarrow \delta_2$.
       done $\leftarrow$ FALSE.
     else
       % Otherwise (1.5) is verified.
       done $\leftarrow$ TRUE.
     end if
   until done = TRUE
2. Return δ_1.

To obtain the distance to uncontrollability with better accuracy, Burke, Lewis, and Overton [3] proposed a trisection variant. The trisection algorithm (Algorithm 2) bounds $\tau(A, B)$ by an interval $[l, u]$ and reduces the length of this interval by a factor of $\frac{2}{3}$ at each iteration. Thus it can compute $\tau(A, B)$ to any desired accuracy in $O(n^6)$ operations which is the cost of Gu's test, as described next.

Algorithm 2 Trisection variant of Algorithm 1

Call: $[l,\, u] \leftarrow$ `Trisection`(A,B,ϵ).
Input: $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times m}$ with $m \le n$, and a tolerance $\epsilon > 0$.
Output: Scalars l and u satisfying $l < \tau(A,B) \le u$ and $u - l < \epsilon$.

1. Initialize the lower bound as $l \leftarrow 0$ and the upper bound as $u \leftarrow \sigma_n([A\ \ B])$.
   repeat
   $\delta_1 \leftarrow l + \frac{2}{3}(u-l)$
   $\delta_2 \leftarrow l + \frac{1}{3}(u-l)$
   Apply Gu's test.
   if (1.4) is verified **then**
   $u \leftarrow \delta_1$.
   else
   % Otherwise (1.5) is verified.
   $l \leftarrow \delta_2$.
   end if
   until $u - l < \epsilon$
2. Return l and u.

2.2. Gu's verification scheme. By means of Gu's test we can numerically verify whether a real pair of solutions to (2.1) exists. Equation (2.1) in Theorem 2.1 implies that there exist nonzero vectors $\binom{x}{y}$, z, $\binom{x}{\hat{y}}$, and $\hat{z}$ such that

$$(2.2a)\qquad (\,A-(\alpha+\beta i)I \quad B\,)\begin{pmatrix}x\\ y\end{pmatrix} = \delta z,\ \begin{pmatrix}A^*-(\alpha-\beta i)I\\ B^*\end{pmatrix} z = \delta\begin{pmatrix}x\\ y\end{pmatrix},$$

$$(2.2b)\qquad (\,A-(\alpha+\eta+\beta i)I \quad B\,)\begin{pmatrix}\hat{x}\\ \hat{y}\end{pmatrix} = \delta\hat{z},\ \begin{pmatrix}A^*-(\alpha+\eta-\beta i)I\\ B^*\end{pmatrix}\hat{z} = \delta\begin{pmatrix}\hat{x}\\ \hat{y}\end{pmatrix}.$$

These equations can be rewritten as

$$(2.3a)\qquad \begin{pmatrix} -\delta I & A-\alpha I & B\\ A^*-\alpha I & -\delta I & 0\\ B^* & 0 & -\delta I\end{pmatrix}\begin{pmatrix}z\\ x\\ y\end{pmatrix} = \beta i\begin{pmatrix}0 & I & 0\\ -I & 0 & 0\\ 0 & 0 & 0\end{pmatrix}\begin{pmatrix}z\\ x\\ y\end{pmatrix}$$

and

$$(2.3b)\qquad \begin{pmatrix} -\delta I & A-(\alpha+\eta)I & B\\ A^*-(\alpha+\eta)I & -\delta I & 0\\ B^* & 0 & -\delta I\end{pmatrix}\begin{pmatrix}\hat{z}\\ \hat{x}\\ \hat{y}\end{pmatrix} = \beta i\begin{pmatrix}0 & I & 0\\ -I & 0 & 0\\ 0 & 0 & 0\end{pmatrix}\begin{pmatrix}\hat{z}\\ \hat{x}\\ \hat{y}\end{pmatrix}.$$

Furthermore using the QR factorization

$$(2.4)\qquad \begin{pmatrix}B\\ -\delta I\end{pmatrix} = \begin{pmatrix}Q_{11} & Q_{12}\\ Q_{21} & Q_{22}\end{pmatrix}\begin{pmatrix}R\\ 0\end{pmatrix}$$

these problems can be reduced to standard eigenvalue problems of size $2n \times 2n$, i.e., the eigenvalues of the pencils in (2.3a) and in (2.3b) are the same as the eigenvalues of the matrices

$$(2.5a)\qquad \begin{pmatrix} A-\alpha I & BQ_{22}-\delta Q_{12}\\ \delta Q_{12}^{-1} & -Q_{12}^{-1}(A^*-\alpha I)Q_{12}\end{pmatrix}$$

and

$$\begin{pmatrix} A-(\alpha+\eta)I & BQ_{22}-\delta Q_{12} \\ \delta Q_{12}^{-1} & -Q_{12}^{-1}(A^*-(\alpha+\eta)I)Q_{12} \end{pmatrix}, \tag{2.5b}$$

respectively. In order for (2.1) to have at least one real solution (α, β), these two matrices must share a common pure imaginary eigenvalue βi. This requires a $2n^2 \times 2n^2$ generalized eigenvalue problem to have a real eigenvalue α (see [6]). For a given δ and η, we check whether the latter generalized eigenvalue problem has any real eigenvalue α. If it does, then we check the existence of a real eigenvalue α for which the matrices (2.5a) and (2.5b) share a common pure imaginary eigenvalue βi. There exists a pair of α and β satisfying (2.1) if and only if this process succeeds.

3. Modified fast verification scheme. It turns out that Gu's verification scheme can be simplified. In this modified scheme the $2n^2 \times 2n^2$ generalized eigenvalue problems whose real eigenvalues are sought in Gu's scheme are replaced by $2n^2 \times 2n^2$ standard eigenvalue problems, and the $2n \times 2n$ standard eigenvalue problems (2.5a) and (2.5b) whose imaginary eigenvalues are sought are replaced by new $2n \times 2n$ standard eigenvalue problems that do not require the computation of QR factorizations.

The simplification of the problem of size $2n^2 \times 2n^2$ is significant, as the inverse of the new matrix of size $2n^2 \times 2n^2$ (whose real eigenvalues are sought) times a vector can be computed in a cheap manner by solving a Sylvester equation of size $2n \times 2n$ with a cost of $O(n^3)$. As a consequence the closest eigenvalue to a given complex point can be computed efficiently by applying shifted inverse iteration or shift-and-invert Arnoldi. We discuss how this idea can be extended to extract all of the real eigenvalues with an average cost of $O(n^4)$ and a worst case cost of $O(n^5)$, reducing the running time of each iteration of the bisection or the trisection algorithm asymptotically.

3.1. New generalized eigenvalue problem. According to (2.2a)

$$y = \frac{1}{\delta} B^* z$$

and the two equations in (2.2a) can be rewritten as

$$\begin{pmatrix} \hat{B} & A-\alpha I \\ A^*-\alpha I & -\delta I \end{pmatrix} \begin{pmatrix} z \\ x \end{pmatrix} = \beta i \begin{pmatrix} & I \\ -I & \end{pmatrix} \begin{pmatrix} z \\ x \end{pmatrix},$$

where $\hat{B} = \frac{BB^*}{\delta} - \delta I$. That is

$$H(\alpha) \begin{pmatrix} z \\ x \end{pmatrix} = \begin{pmatrix} -(A^*-\alpha I) & \delta I \\ \hat{B} & A-\alpha I \end{pmatrix} \begin{pmatrix} z \\ x \end{pmatrix} = \beta i \begin{pmatrix} z \\ x \end{pmatrix}. \tag{3.1a}$$

Similarly

$$H(\alpha+\eta) \begin{pmatrix} \hat{z} \\ \hat{x} \end{pmatrix} = \begin{pmatrix} -(A^*-(\alpha+\eta)I) & \delta I \\ \hat{B} & A-(\alpha+\eta)I \end{pmatrix} \begin{pmatrix} \hat{z} \\ \hat{x} \end{pmatrix} = \beta i \begin{pmatrix} \hat{z} \\ \hat{x} \end{pmatrix}. \tag{3.1b}$$

Both of the eigenvalue problems above are Hamiltonian, i.e., $JH(\alpha)$ and $JH(\alpha+\eta)$ are Hermitian where

$$J = \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix} \tag{3.2}$$

with $n \times n$ blocks. The Hamiltonian property implies that the matrices $H(\alpha+\eta)$ and $-H(\alpha+\eta)^*$ have the same set of eigenvalues. For $H(\alpha)$ and $H(\alpha+\eta)$ or equivalently $H(\alpha)$ and $-H(\alpha+\eta)^*$ to share a common pure eigenvalue βi, the following matrix equation

$$\begin{pmatrix} -(A^* - \alpha I) & \delta I \\ \hat{B} & A - \alpha I \end{pmatrix} X + X \begin{pmatrix} -(A^* - (\alpha+\eta) I) & \delta I \\ \hat{B} & A - (\alpha+\eta) I \end{pmatrix}^* = 0 \tag{3.3}$$

or equivalently

$$\begin{pmatrix} -A^* & \delta I \\ \hat{B} & A \end{pmatrix} X + X \begin{pmatrix} -(A - \eta I) & \hat{B} \\ \delta I & A^* - \eta I \end{pmatrix} = \alpha \left(\begin{pmatrix} -I & 0 \\ 0 & I \end{pmatrix} X + X \begin{pmatrix} -I & 0 \\ 0 & I \end{pmatrix} \right) \tag{3.4}$$

must have a nonzero solution $X \in \mathbb{C}^{2n \times 2n}$. Partition $X = \left(\begin{smallmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{smallmatrix}\right)$, and let $\mathbf{vec}(X)$ denote the vector formed by stacking the column vectors of X. We will use the following properties of Kronecker products:

$$\mathbf{vec}(AX) = (I \otimes A)\mathbf{vec}(X), \ \mathbf{vec}(XA) = (A^T \otimes I)\mathbf{vec}(X).$$

Now we can rewrite (3.4) as

$$\begin{pmatrix} -A_1^* - A_2^T & \delta I & \delta I & 0 \\ B_2^T & -A_1^* + \bar{A}_2 & 0 & \delta I \\ B_1 & 0 & A_1 - A_2^T & \delta I \\ 0 & B_1 & B_2^T & A_1 + \bar{A}_2 \end{pmatrix} \begin{pmatrix} \mathbf{vec}(X_{11}) \\ \mathbf{vec}(X_{12}) \\ \mathbf{vec}(X_{21}) \\ \mathbf{vec}(X_{22}) \end{pmatrix} = \begin{pmatrix} -2\alpha \mathbf{vec}(X_{11}) \\ 0 \\ 0 \\ 2\alpha \mathbf{vec}(X_{22}) \end{pmatrix}, \tag{3.5}$$

where $A_1 = I \otimes A$, $A_2 = (A - \eta I) \otimes I$, $B_1 = I \otimes \hat{B}$, $B_2 = \hat{B} \otimes I$, and $\bar{A}_2$ denotes the matrix obtained by taking the complex conjugate of A_2 entrywise.

The $(1,2)$, $(2,1)$ entries of both sides of (3.4) lead to

$$\begin{pmatrix} B_2^T & \delta I & -A_1^* + \bar{A}_2 & 0 \\ B_1 & \delta I & 0 & A_1 - A_2^T \end{pmatrix} \begin{pmatrix} \mathbf{vec}(X_{11}) \\ \mathbf{vec}(X_{22}) \\ \mathbf{vec}(X_{12}) \\ \mathbf{vec}(X_{21}) \end{pmatrix} = 0.$$

We then have

$$\begin{pmatrix} \mathbf{vec}(X_{12}) \\ \mathbf{vec}(X_{21}) \end{pmatrix} = - \begin{pmatrix} -A_1^* + \bar{A}_2 & 0 \\ 0 & A_1 - A_2^T \end{pmatrix}^{-1} \begin{pmatrix} B_2^T & \delta I \\ B_1 & \delta I \end{pmatrix} \begin{pmatrix} \mathbf{vec}(X_{11}) \\ \mathbf{vec}(X_{22}) \end{pmatrix} \tag{3.6}$$

under the assumption that A does not have two eigenvalues that differ by η, in which case the matrix $A_1 - A_2^T$ is invertible and therefore the inverted matrix in (3.6) exists. This assumption is generically satisfied in practice (numerical troubles that occur when η is small are discussed in section 5). On the other hand the $(1,1)$, $(2,2)$ entries

of both sides of (3.4) give

$$\begin{pmatrix} -A_1^* - A_2^T & 0 & \delta I & \delta I \\ 0 & A_1 + \bar{A}_2 & B_1 & B_2^T \end{pmatrix} \begin{pmatrix} \mathbf{vec}(X_{11}) \\ \mathbf{vec}(X_{22}) \\ \mathbf{vec}(X_{12}) \\ \mathbf{vec}(X_{21}) \end{pmatrix} = 2\alpha \begin{pmatrix} -I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{vec}(X_{11}) \\ \mathbf{vec}(X_{22}) \\ \mathbf{vec}(X_{12}) \\ \mathbf{vec}(X_{21}) \end{pmatrix},$$

which can be simplified with (3.6) to

$$\left[\begin{pmatrix} -A_1^* - A_2^T & 0 \\ 0 & A_1 + \bar{A}_2 \end{pmatrix} - \begin{pmatrix} \delta I & \delta I \\ B_1 & B_2^T \end{pmatrix} \begin{pmatrix} -A_1^* + \bar{A}_2 & 0 \\ 0 & A_1 - A_2^T \end{pmatrix}^{-1} \begin{pmatrix} B_2^T & \delta I \\ B_1 & \delta I \end{pmatrix} \right] \begin{pmatrix} \mathbf{vec}(X_{11}) \\ \mathbf{vec}(X_{22}) \end{pmatrix} = 2\alpha \begin{pmatrix} -I & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \mathbf{vec}(X_{11}) \\ \mathbf{vec}(X_{22}) \end{pmatrix},$$

i.e.,

$$\mathcal{A}v = \alpha v, \tag{3.7}$$

where

$$\mathcal{A} = \frac{1}{2} \left[\begin{pmatrix} A_1^* + A_2^T & 0 \\ 0 & A_1 + \bar{A}_2 \end{pmatrix} - \begin{pmatrix} -\delta I & -\delta I \\ B_1 & B_2^T \end{pmatrix} \begin{pmatrix} -A_1^* + \bar{A}_2 & 0 \\ 0 & A_1 - A_2^T \end{pmatrix}^{-1} \begin{pmatrix} B_2^T & \delta I \\ B_1 & \delta I \end{pmatrix} \right]. \tag{3.8}$$

For the verification of a pair α and β satisfying (2.1), we first solve the eigenvalue problem (3.7). If there exists a real eigenvalue α of this problem such that the matrices $H(\alpha)$ and $H(\alpha + \eta)$ share a common imaginary eigenvalue, then the verification succeeds.

3.2. Inverse iteration. The eigenvalue problem in (3.7) is a simplified version of the generalized eigenvalue problem in [6]. This is a problem of finding the real eigenvalues of a nonsymmetric matrix. The implementation[2] of the trisection algorithm of [3] uses the MATLAB function `eig` to compute the eigenvalues of that generalized eigenvalue problem with a cost of $O(n^6)$ and therefore does not exploit the fact that we need only the real eigenvalues of the generalized problem. In section 3.3 we discuss two strategies to extract the real eigenvalues of a given matrix $\mathcal{X}$ that is preferable to `eig` when the closest eigenvalue of $\mathcal{X}$ to a given point can be obtained efficiently.

In this section we show how one can compute the closest eigenvalue of $\mathcal{A}$ to a given point in the complex plane in $O(n^3)$ time. This is due to the fact that given a shift ν and a vector $u \in \mathbb{C}^{2n^2}$, the multiplication $(\mathcal{A} - \nu I)^{-1}u$ can be performed by solving a Sylvester equation of size $2n \times 2n$ which is derived next. Therefore, shifted inverse iteration or shift-and-invert Arnoldi can locate the closest eigenvalue efficiently.

[2] http://www.cs.nyu.edu/faculty/overton/software/uncontrol/.

3.2.1. Computing $\mathcal{A}^{-1}u$. We first derive the Sylvester equation whose solution yields $v = \mathcal{A}^{-1}u$, where $u = \binom{u_1}{u_2}$, $v = \binom{v_1}{v_2}$, and $u_1, u_2, v_1, v_2 \in \mathbb{C}^{n^2}$. We can also write

$$\mathcal{A}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}. \tag{3.9}$$

Let

$$w = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} -A_1^* + \bar{A}_2 & 0 \\ 0 & A_1 - A_2^T \end{pmatrix}^{-1} \begin{pmatrix} B_2^T & \delta I \\ B_1 & \delta I \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}. \tag{3.10}$$

Then (3.9) can be rewritten as

$$\begin{pmatrix} A_1^* + A_2^T & 0 \\ 0 & A_1 + \bar{A}_2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} - \begin{pmatrix} -\delta I & -\delta I \\ B_1 & B_2^T \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = 2 \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}. \tag{3.11}$$

Equations (3.10) and (3.11) can then be combined into one linear system

$$\begin{pmatrix} A_1^* + A_2^T & \delta I & \delta I & 0 \\ B_2^T & A_1^* - \bar{A}_2 & 0 & \delta I \\ B_1 & 0 & -A_1 + A_2^T & \delta I \\ 0 & -B_1 & -B_2^T & A_1 + \bar{A}_2 \end{pmatrix} \begin{pmatrix} v_1 \\ w_1 \\ w_2 \\ v_2 \end{pmatrix} = 2 \begin{pmatrix} u_1 \\ 0 \\ 0 \\ u_2 \end{pmatrix}, \tag{3.12}$$

which is analogous to (3.5). By introducing vector forms

$$u = \begin{pmatrix} \mathbf{vec}(U_1) \\ \mathbf{vec}(U_2) \end{pmatrix}, \; v = \begin{pmatrix} \mathbf{vec}(V_1) \\ \mathbf{vec}(V_2) \end{pmatrix}, \; w = \begin{pmatrix} \mathbf{vec}(W_1) \\ \mathbf{vec}(W_2) \end{pmatrix},$$

we get a matrix equation similar to (3.4),

$$\begin{pmatrix} A^* & \delta I \\ \hat{B} & -A \end{pmatrix} Z + Z \begin{pmatrix} A - \eta I & \hat{B} \\ \delta I & -A^* + \eta I \end{pmatrix} = 2 \begin{pmatrix} U_1 & 0 \\ 0 & -U_2 \end{pmatrix}, \tag{3.13}$$

where

$$Z = \begin{pmatrix} V_1 & W_1 \\ W_2 & V_2 \end{pmatrix}. \tag{3.14}$$

Equation (3.13) is a $2n \times 2n$ Sylvester equation. By using a Sylvester equation solver (such as the LAPACK routine `dtrsyl` [1]) we can solve for Z at $O(n^3)$ cost and thus obtain $v = \mathcal{A}^{-1}u$.

3.2.2. Computing $(\mathcal{A} - \nu I)^{-1}u$. The derivation of the Sylvester equation for the multiplication $\mathcal{A}^{-1}u$ easily extends to the multiplication $(\mathcal{A} - \nu I)^{-1}u$ for a given shift ν. We alternatively rewrite the multiplication as

$$(\mathcal{A} - \nu I)\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \tag{3.15}$$

and introduce $w = \binom{w_1}{w_2}$ as in the previous section. We end up with

$$\begin{pmatrix} A_1^* + A_2^T - 2\nu I & \delta I & \delta I & 0 \\ B_2^T & A_1^* - \bar{A}_2 & 0 & \delta I \\ B_1 & 0 & -A_1 + A_2^T & \delta I \\ 0 & -B_1 & -B_2^T & A_1 + \bar{A}_2 - 2\nu I \end{pmatrix} \begin{pmatrix} v_1 \\ w_1 \\ w_2 \\ v_2 \end{pmatrix} = 2 \begin{pmatrix} u_1 \\ 0 \\ 0 \\ u_2 \end{pmatrix}, \tag{3.16}$$

which is analogous to (3.12). In terms of a matrix equation, we obtain

$$\begin{pmatrix} A^* - \nu I & \delta I \\ \hat{B} & -A + \nu I \end{pmatrix} Z + Z \begin{pmatrix} A - (\eta + \nu) I & \hat{B} \\ \delta I & -A^* + (\eta + \nu) I \end{pmatrix} = 2 \begin{pmatrix} U_1 & 0 \\ 0 & -U_2 \end{pmatrix}, \tag{3.17}$$

where Z is as defined in (3.14). Equation (3.17) is identical to (3.13) except that A is replaced by $A - \nu I$ in (3.13) and thus can be solved at $O(n^3)$ cost.

3.3. Real eigenvalue searching strategies. In this section we seek the real eigenvalues of a given matrix $\mathcal{X} \in \mathbb{C}^{q \times q}$. The iterative methods here are preferable to the standard ways of computing eigenvalues such as the QR algorithm when $(\mathcal{X} - \nu I)^{-1} u$ for a given shift $\nu \in \mathbb{R}$ and a given vector $u \in \mathbb{C}^q$ is efficiently computable. In particular, as discussed in the previous section, this is the case for $\mathcal{A}$.

Throughout this section we will assume the existence of a reliable implementation of the shifted inverse iteration or a shift-and-invert Arnoldi method that returns the closest eigenvalue to a given shift accurately. In practice we make use of the MATLAB function `eigs` (based on ARPACK [9, 10]). Additionally, we assume that an upper bound, D, on the norm of $\mathcal{X}$ is available and therefore we know that all of the real eigenvalues lie in the interval $[-D, D]$. A straightforward approach would be to partition the interval $[-D, D]$ into equal subintervals and find the closest eigenvalue to the midpoint of each interval. This approach must work as long as the subintervals are chosen small enough. Nevertheless, partitioning $[-D, D]$ into very fine subintervals is not desirable, since this will require an excessive number of closest eigenvalue computations. Next we present two viable approaches that are both reliable and efficient.

3.3.1. Adaptive progress. The first approach we present here is rather brute-force. In addition to the existence of an upper bound D on the norm of $\mathcal{X}$, we assume that a positive number

$$d \leq \min_{\lambda_i, \lambda_j : \text{distinct eigenvalues}} |\lambda_i - \lambda_j| \tag{3.18}$$

is known *a priori*. We start from the right endpoint D as our initial shift. At each iteration we compute the closest eigenvalue to the current shift and decrement the current shift by an amount depending on the distance from the computed eigenvalue to the shift. We keep decrementing the shift until we reach the left endpoint.

The way the shift ν is updated depends on the closest eigenvalue λ that is found. If λ is real and already discovered, then λ must be larger than ν. In this case there is no real eigenvalue in the interval $(\nu - (\lambda - \nu), \nu]$. Additionally there is no real eigenvalue in the interval $(\lambda - d, \lambda]$. The corresponding update rule in Algorithm 3 combines these two conditions. When λ is real and not discovered, the shift ν is $\lambda - d$. Finally when the closest eigenvalue is not real, the new shift is set to the leftmost of the intersection points of two circles with the real line. One of the circles is centered at ν and has radius $|\lambda - \nu|$. The second circle is centered at λ and has radius d.

In Figure 3.1 the progress of Algorithm 3 on an example is shown. The algorithm iterates 10 times to investigate the part of the real axis where the eigenvalues are known to lie. In particular notice that the algorithm locates the same eigenvalue near the real axis at the second, third, and fourth iterations and another on the real axis at the fifth, sixth, and seventh iterations. Locating the same eigenvalue a few times is a deficiency of this algorithm.

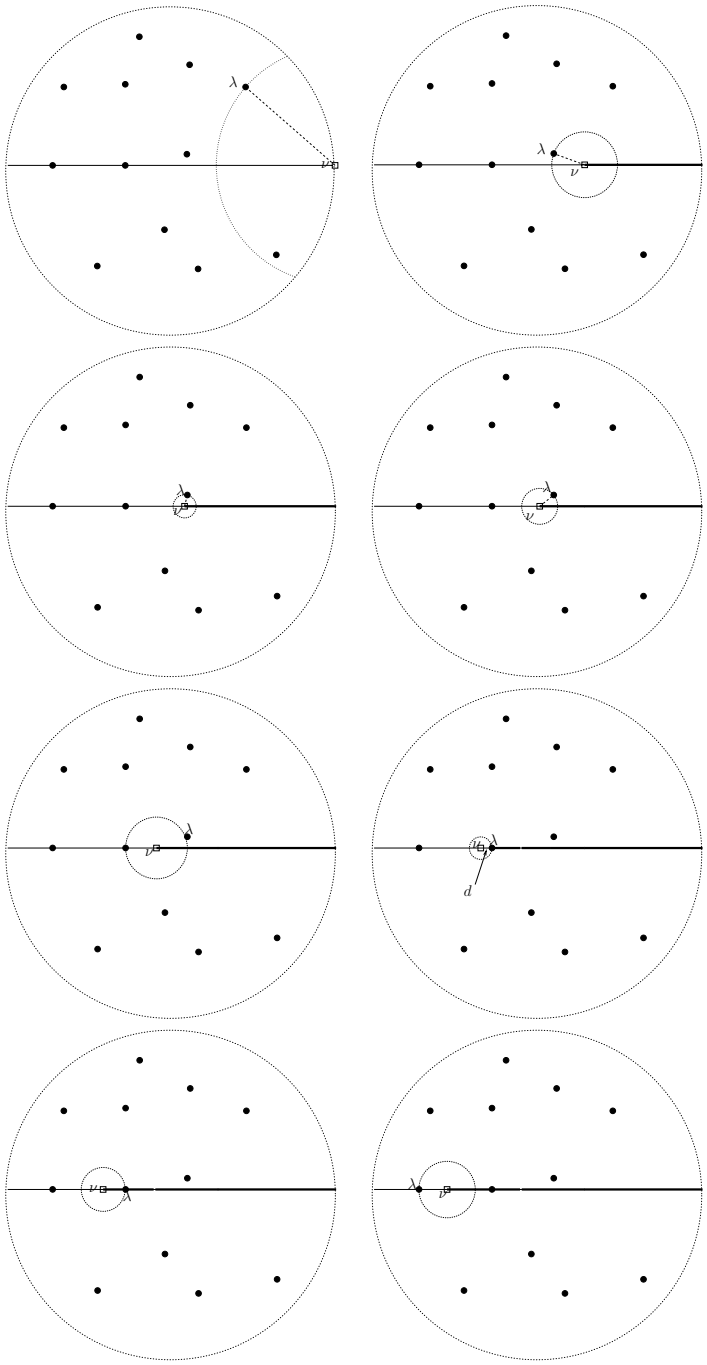

FIG. 3.1. *The first eight iterations (top leftmost is the first iteration; iteration numbers increase from left to right and top to bottom) of the adaptive progress algorithm on an example are displayed. Black dots denote the eigenvalues. Squares mark the location of the shift ν. The closest eigenvalue to ν is denoted by λ. The part of the real axis already investigated is marked by a thicker line. Iterations $2, 3, 4$ locate the same eigenvalue close to the real axis and iterations $5, 6, 7$ locate the same real eigenvalue repeatedly. Little progress is achieved in moving the shift toward the left during these iterations. When an undiscovered real eigenvalue is located, the next shift is obtained by subtracting d, a lower bound on the distance of the closest two eigenvalues, from the real eigenvalue.*

Algorithm 3 Adaptive progress real eigenvalue search algorithm

Call:	$\Lambda \leftarrow$ `Adaptive_Progress`($\mathcal{X}$,D,d).
Input:	$\mathcal{X} \in \mathbb{C}^{q\times q}$, D, an upper bound on the norm of $\mathcal{X}$, and d, a lower bound for the distance between the closest two eigenvalues of $\mathcal{X}$.
Output:	$\Lambda \in \mathbb{R}^l$ with $l \leq q$ containing all of the real eigenvalues of $\mathcal{X}$ in the interval $[-D, D]$.

1. Initially set the shift $\nu \leftarrow D$ and the vector of real eigenvalues $\Lambda \leftarrow [\,]$.
   while $\nu \geq -D$ **do**
     Compute the closest eigenvalue λ to the shift ν.
     if λ is real **then**
       if $\lambda \in \Lambda$ **then**
         % λ is real and already discovered.
         $\nu \leftarrow \nu - \max(|\lambda - \nu|, d - |\lambda - \nu|)$.
       else
         % λ is real but not discovered yet.
         Add λ to Λ.
         $\nu \leftarrow \lambda - d$.
       end if
     else
       % Otherwise λ is not purely real. Choose the leftmost
       % intersection point of the circle centered at ν and with
       % radius $|\nu - \lambda|$ and the circle centered at λ and with
       % radius d with the real line as the new shift.
       if $d \geq \operatorname{Im}\lambda$ **then**
         % Both of the circles intersect the real line.
         $\nu \leftarrow \min(\operatorname{Re}\lambda - \sqrt{d^2 - \operatorname{Im}\lambda^2}, \nu - |\nu - \lambda|)$.
       else
         % Only the circle centered at ν intersects the real line.
         $\nu \leftarrow \nu - |\nu - \lambda|$.
       end if
     end if
   end while
2. Return the real eigenvalue list Λ.

From the description of the algorithm it is not clear whether it terminates. The next theorem shows that the adaptive progress algorithm indeed terminates.

THEOREM 3.1. *Let the shift of Algorithm* 3 *at a given iteration be* ν. *The next shift will be no larger than* $\nu - \frac{d}{2}$. *Thus the number of closest eigenvalue computations is* $O(\frac{D}{d})$.

Proof. Clearly when the closest eigenvalue λ is real, the shift is decremented by at least $d/2$. Therefore let us focus on the case when the eigenvalue λ is not real.

We will find a lower bound for the progress h such that the next shift is $\nu - h$. When only the circle centered at ν intersects the real line (i.e., the imaginary part of λ is greater than d), it is apparent from Figure 3.2c that the distance $|\lambda - \nu|$ is greater than or equal to d. Since the next shift is set to the intersection point $\nu - |\lambda - \nu|$, we have $h = |\lambda - \nu| \geq d$. When both of the circles intersect the real line, as in Figure 3.2a and Figure 3.2b, the progress h is the maximum of the lengths of the line segment

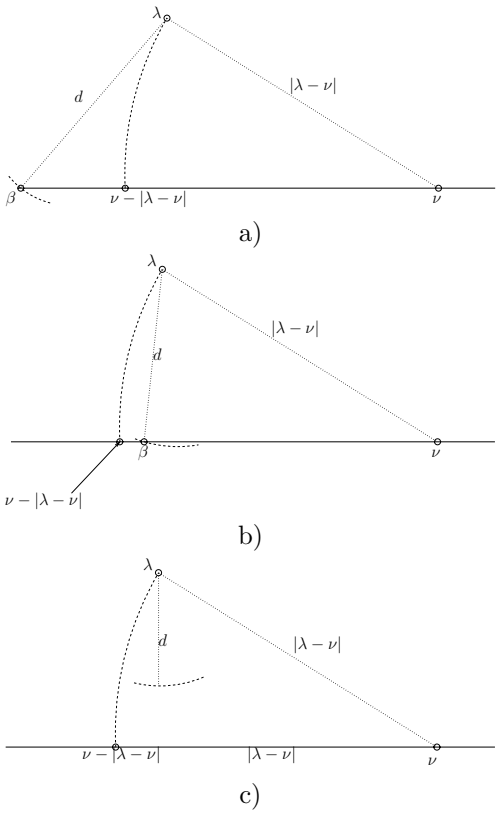

FIG. 3.2. *Three possibilities for the adaptive progress algorithm when the closest eigenvalue λ to the shift ν is not real. The circular arcs are arcs of the circle centered at ν with radius $|\lambda-\nu|$ and the circle centered at λ with radius d. The point β is the intersection point of the second circle with the real line.* a) *Both of the circles intersect the real line, and the circle centered at λ intersects the real line further to the left.* b) *Both of the circles intersect the real line, and the circle centered at ν intersects the real line further to the left.* c) *Only the circle centered at ν intersects the real line.*

from ν to λ and the line segment from ν to β. In both cases, since in the triangle with vertices ν, β and λ, the length of the third edge from β to λ is d, the triangular inequality yields

$$\max(|\nu - \lambda|, |\nu - \beta|) \geq \frac{d}{2}.$$

As the length of the interval to be searched is $2D$, the maximum number of closest eigenvalue computations is bounded by $\frac{4D}{d}$. ∎

We point out a few disadvantages of the adaptive progress approach. The most obvious is the need for a lower bound on the distance between the closest two eigenvalues of $\mathcal{X}$. For reliability, one needs to set d small. The consequence of choosing d small, however, is too many closest eigenvalue computations. A second disadvantage of the adaptive progress approach is that once it detects a new real eigenvalue, it will typically continue to converge to the same eigenvalue with the next few shifts that are close to the eigenvalue. Finally, when an upper bound on $\|\mathcal{X}\|$ is not available, in a robust implementation D must be chosen large, and clearly this degrades the

performance of the algorithm.

3.3.2. Divide and conquer. As an alternative to the adaptive progress approach it is possible to apply a divide and conquer algorithm. Unlike the adaptive progress approach, the divide and conquer algorithm does not require the knowledge of a lower bound on the distance between the closest two eigenvalues of $\mathcal{X}$. In addition it chooses shifts that are away from the computed eigenvalues in order to avoid the discovery of the same eigenvalue too many times (to be precise, each eigenvalue can be discovered at most three times). Even though an upper bound D on the norm of $\mathcal{X}$ is required and may not be available in general, in practice we can choose D very large so that the interval $[-D, D]$ contains all of the real eigenvalues and, as we discuss, this affects the number of closest eigenvalue computations insignificantly. The factor most affecting the efficiency of the algorithm is the eigenvalue distribution.

Algorithm 4 Divide and conquer real eigenvalue search algorithm

Call:	$\Lambda \leftarrow$ `Divide_And_Conquer`$(\mathcal{X},L,U)$.
Input:	$\mathcal{X} \in \mathbb{C}^{q\times q}$, a lower bound L for the smallest real eigenvalue desired and an upper bound U for the largest real eigenvalue desired.
Output:	$\Lambda \in \mathbb{R}^l$ with $l \leq q$ containing all of the real eigenvalues of $\mathcal{X}$ in the interval $[L, U]$.

1. Set the shift $\nu \leftarrow \frac{(U+L)}{2}$.
2. Compute the eigenvalue λ closest to the shift ν.

if $U - L < 2|\lambda - \nu|$ **then**
  % Base case: there is no real eigenvalue in the interval $[L, U]$.
  Return [ ].
else
  % Recursive case: Search the left and right intervals.
  $\Lambda_L \leftarrow$ `Divide_And_Conquer`$(\mathcal{X},L,\nu - |\lambda - \nu|)$
  $\Lambda_R \leftarrow$ `Divide_And_Conquer`$(\mathcal{X},\nu + |\lambda - \nu|,U)$
  % Combine all of the real eigenvalues.
  if λ is real **then**
    Return $\lambda \cup \Lambda_L \cup \Lambda_R$.
  else
    Return $\Lambda_L \cup \Lambda_R$.
  end if
end if

In this approach, given an interval $[L, U]$ we compute the eigenvalue of $\mathcal{X}$ closest to the midpoint of the interval $\nu = \frac{U+L}{2}$. If the modulus of the difference between the computed eigenvalue λ and the midpoint ν is greater than half of the length of the interval, then we terminate. Otherwise we apply the same procedure to the subintervals $[L, \nu - |\lambda - \nu|]$ and $[\nu + |\lambda - \nu|, U]$. Initially we apply the algorithm to the whole interval $[-D, D]$.

Figure 3.3 illustrates the first six iterations of the divide and conquer algorithm on the same example used in Figure 3.1. The divide and conquer algorithm completes the investigation of the real interval where the real eigenvalues reside after iterating 7 times as opposed to the 10 iterations required by the adaptive progress algorithm.

For reliability the parameter D must be chosen large. Suppose all the eigenvalues

are contained in the disk of radius D' with $D' \ll D$. To discover that there is no real eigenvalue in the interval $[D', D]$, at most two extra closest eigenvalue computations are required. If the first shift tried in the interval $[D', D]$ is closer to D' than D, then the distance from the closest eigenvalue to this shift may be less than half the length of the interval $[D', D]$, so a second closest eigenvalue computation may be needed. Otherwise the interval $[D', D]$ will be investigated in one iteration. Similar remarks hold for the interval $[-D, -D']$. However, the larger choices of D may slightly increase or decrease the number of shifts required to investigate $[-D', D']$. The important point is that regardless of how large D is compared to the radius of the smallest disk containing the eigenvalues, the cost is limited to approximately four extra iterations.

Next we show that the number of closest eigenvalue computations cannot exceed $2q + 1$ (recall that $\mathcal{X} \in \mathbb{C}^{q \times q}$).

THEOREM 3.2 (worst case for Algorithm 4). *The number of closest eigenvalue computations made by Algorithm* 3.2 *is no more than* $2q + 1$.

Proof. We can represent the progress of the algorithm by a full binary tree, i.e., a tree with each node having either two children or no children. Each node of the tree corresponds to an interval. The root of the tree corresponds to the whole interval $[-D, D]$. At each iteration of the algorithm the interval under consideration is either completely investigated or replaced by two disjoint subintervals that need to be investigated. In the first case, the node corresponding to the current interval is a leaf. In the second case, the node has two children, one for each of the subintervals.

We claim that the number of leaves in this tree cannot exceed $q+1$. The intervals corresponding to the leaves are disjoint. Each such interval has a closest left interval (except the leftmost interval) and a closest right interval (except the rightmost interval) represented by two of the leaves in the tree. Each interval is separated from the closest one on the left by the part of a disk on the real axis in which an eigenvalue lies, and similarly for the closest interval on the right. Since the matrix $\mathcal{X}$ has q eigenvalues, there can be at most q separating disks and therefore at most $q + 1$ disjoint intervals represented by the leaves of the tree. A full binary tree with $q+1$ leaves has q internal nodes. Therefore, the total number of the nodes in the tree, which is the same as the number of closest eigenvalue computations, cannot exceed $2q + 1$. ∎

The upper bound $2q + 1$ on the worst case performance of the algorithm is tight, as illustrated by the following example. Consider a matrix with the real eigenvalues $\frac{2^{j-1}-1}{2^{j-1}}$, $j = 1, \ldots, q$, and suppose we search over the interval $[-1, 1]$. Clearly, the algorithm discovers each eigenvalue twice except the largest one, which it discovers three times (assuming that when there are two eigenvalues equally close to a midpoint, the algorithm locates the eigenvalue on the right). Therefore, the total number of closest eigenvalue computations is $2q + 1$.

Next we aim to show that the average case performance of the algorithm is much better than the worst case. First we note the following elementary result that is an immediate consequence of the fact that the square-root function is strictly concave.

LEMMA 3.3. *Given l positive distinct integers $k_1, k_2, \ldots, k_l$ and l real numbers $p_1, p_2, \ldots, p_l \in (0, 1)$ such that $\sum_{j=1}^{l} p_j = 1$, the inequality*

$$\sqrt{\sum_{j=1}^{l} p_j k_j} > \sum_{j=1}^{l} p_j \sqrt{k_j} \tag{3.19}$$

holds.

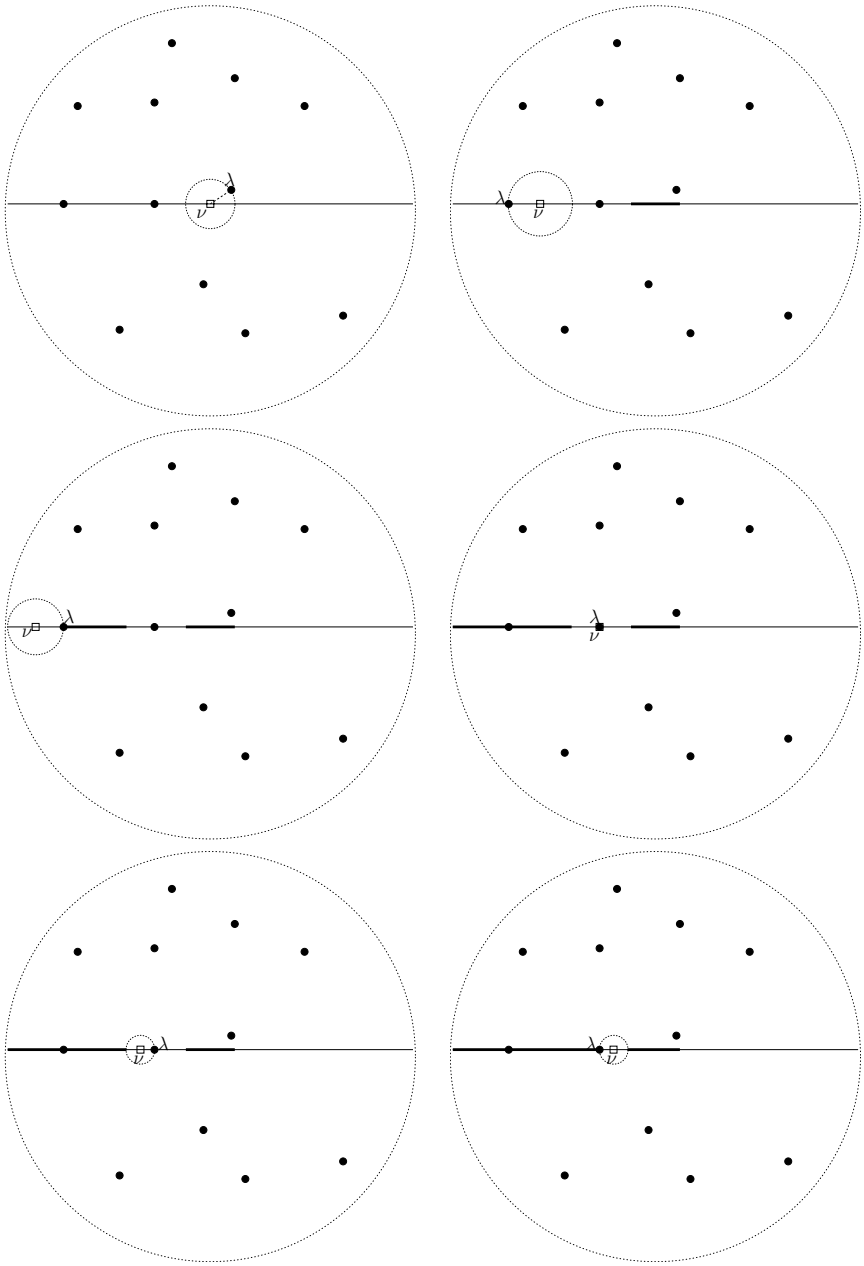

FIG. 3.3. *First six iterations of the divide and conquer algorithm on the same example used in Figure* 3.1.

In the average case analysis we let the eigenvalues of $\mathcal{X}$, say $\xi_1, \xi_2, \ldots, \xi_q$, vary. We assume that the eigenvalues are independently selected from a uniform distribution inside the circle centered at the origin with radius μ. We use Algorithm 4 to compute the real eigenvalues lying inside the circle of radius $D = 1 \leq \mu$ (the value of the radius D is irrelevant for the average case analysis as discussed below; we choose $D = 1$ for simplicity). In Table 3.1 the random variables and the probability density functions

TABLE 3.1
Notation for Theorem 3.4.

X	:	Number of iterations performed by Algorithm 4.
N	:	Number of eigenvalues lying inside the unit circle.
H	:	Modulus of the eigenvalue closest to the origin.
X_l	:	Number of iterations performed by Algorithm 4 on the left interval $[-1, -H]$.
X_r	:	Number of iterations performed by Algorithm 4 on the right interval $[H, 1]$.
N_l	:	Number of eigenvalues lying inside the left circle centered at $-(1+H)/2$ with radius $(1-H)/2$.
$h(H\|N=j)$	:	The probability density function of the variable H given there are j eigenvalues inside the unit circle.
$g_l(N_l\|N=j, H=\beta)$	:	The probability density function of the variable N_l given there are j eigenvalues inside the unit circle and the smallest of the moduli of the eigenvalues is β.

referenced by the proof of the next theorem are summarized.

The quantity we are interested in is $E(X|N=j)$, the expected number of iterations required by Algorithm 4 given that there are j eigenvalues inside the unit circle.

We list a few observations.

- *The eigenvalues $\omega_1, \omega_2, \ldots, \omega_j$ contained in the circle of radius $D = 1$ are uniformly distributed and mutually independent:* This is a simple consequence of the assumption that the eigenvalues are selected from the uniform distribution mutually independently. Let the eigenvalues inside the unit circle be $\xi_{i_1}, \xi_{i_2}, \ldots, \xi_{i_j}$ with $i_1 < i_2 < \cdots < i_j$. We associate ω_k with the location of the kth smallest indexed eigenvalue inside the unit circle, i.e., $\omega_k = \xi_{i_k}$. Let C_1 denote the unit circle. The variable ω_k is uniformly distributed because

$$\begin{aligned} & p(\omega_k | j \text{ of } \xi_1, \ldots, \xi_q \in C_1) \\ &= \sum_{i_1,\ldots,i_j} p(\xi_{i_1}, \ldots, \xi_{i_j} \in C_1 | j \text{ of } \xi_1, \ldots, \xi_q \in C_1) p(\omega_k | \xi_{i_1}, \ldots, \xi_{i_j} \in C_1) \\ &= \sum_{i_1,\ldots,i_j} \binom{q}{j}^{-1} p(\omega_k | \xi_{i_k} \in C_1) \\ &= \sum_{i_1,\ldots,i_j} \binom{q}{j}^{-1} \frac{1}{\pi} = \frac{1}{\pi}. \end{aligned}$$

  Above the summation is over the subsets of $\xi_1, \xi_2, \ldots, \xi_q$ consisting of j elements. Similarly we can show that for $k \neq l$,

$$p(\omega_k, \omega_l | j \text{ of } \xi_1, \ldots, \xi_q \in C_1) = \frac{1}{\pi^2}.$$

  Therefore the variables $\omega_1, \omega_2, \ldots, \omega_j$ are mutually independent.
- *The eigenvalues $\phi_1, \phi_2, \ldots, \phi_{j-1}$ inside the unit circle but outside the circle of radius H are uniformly distributed and mutually independent:* Suppose $\omega_{i_1}, \omega_{i_2}, \ldots, \omega_{i_{j-1}}$ with $i_1 < i_2 < \cdots < i_{j-1}$ are the eigenvalues inside the desired area. When we map ω_{i_k} to ϕ_k, the argument above applies to prove the uniformity and mutual independence of the variables $\phi_1, \phi_2, \ldots, \phi_{j-1}$.

- *Given c eigenvalues $\vartheta_1, \vartheta_2, \ldots, \vartheta_c$ inside the left circle with radius $\frac{1-H}{2}$ centered at $(\frac{-(1+H)}{2}, 0)$, each eigenvalue is uniformly distributed and mutually independent:* This again follows from the arguments above by mapping ϕ_{i_k} to ϑ_k, where $\phi_{i_1}, \phi_{i_2}, \ldots, \phi_{i_c}$ are the eigenvalues inside the desired region with $i_1 < i_2 < \cdots < i_c$.
- *Assuming the number of eigenvalues contained in the circle of radius D is fixed, the expected number of iterations by the algorithm does not depend on the radius D:* Consider the variables $\hat{\omega}_1$ denoting the locations of the j eigenvalues all inside the circle of radius D_1, and $\hat{\omega}_2 = \frac{D_2\hat{\omega}_1}{D_1}$ denoting the locations of the j eigenvalues inside the circle of radius D_2. Let us denote the number of iterations by Algorithm 4 with input $\hat{\omega}_1$ over the interval $[-D_1, D_1]$ by $X_1(\hat{\omega}_1)$ and the number of iterations with input $\hat{\omega}_2$ over the interval $[-D_2, D_2]$ by $X_2(\hat{\omega}_2)$. It immediately follows that $X_1(\hat{\omega}_1) = X_1(\frac{D_1\hat{\omega}_2}{D_2}) = X_2(\hat{\omega}_2)$. By exploiting this equality we can deduce $E(X_1|N_1 = j) = E(X_2|N_2 = j)$,

$$\begin{aligned} E(X_1|N_1 = j) &= \int_{C_{D_1}} X_1(\hat{\omega}_1)p(\hat{\omega}_1)\, d\hat{\omega}_1 \\ &= \left(\frac{1}{\pi D_1^2}\right)^j \int_{C_{D_1}} X_1(\hat{\omega}_1)\, d\hat{\omega}_1 \\ &= \left(\frac{1}{\pi D_1^2}\right)^j \int_{C_{D_2}} X_1\left(\frac{D_1\hat{\omega}_2}{D_2}\right) \frac{D_1^{2j}}{D_2^{2j}} d\hat{\omega}_2 \\ &= \int_{C_{D_2}} X_2(\hat{\omega}_2)p(\hat{\omega}_2)\, d\hat{\omega}_2 \\ &= E(X_2|N_2 = j), \end{aligned}$$

  where N_1 and N_2 are the number of eigenvalues inside the circle of radius C_{D_1} of radius D_1 and the circle of radius C_{D_2} of radius D_2, respectively. Note that the eigenvalues inside both the circle C_{D_1} and the circle C_{D_2} are uniformly distributed and independent, as we discussed above.

By combining these remarks we conclude the equality $E(X|N = j) = E(X_l|N_l = j)$, since the eigenvalues are uniformly distributed and independent inside the circles, and the sizes of the circles do not affect the expected number of iterations given that there are j eigenvalues inside the circles.

The next theorem establishes a recurrence equation for $E(X|N = j)$ in terms of $E(X|N = k)$, $k = 0 \ldots j-1$. Using the recurrence equation we will show $E(X|N = j) = O(\sqrt{j})$ by induction. For convenience let us use the shorthand notation $E_j(X)$ for $E(X|N = j)$.

THEOREM 3.4. *Suppose the eigenvalues of the input matrices of size q are chosen from a uniform distribution independently inside the circle of radius μ and Algorithm 4 is run over the interval $[-1, 1]$. The quantity $E_j(X)$ can be characterized by the recurrence equation*

$$E_0(X) = 1 \tag{3.20}$$

and for all $0 < j < q$

$$E_j(X) = 2F_{j-1}(X) + 1, \tag{3.21}$$

where $F_{j-1}(X)$ is a linear combination of the expectations $E_0(X), \ldots, E_{j-1}(X)$,

$$F_{j-1}(X) = \int_0^1 \left(\sum_{k=0}^{j-1} E_k(X) g_l(N_l = k | N = j, H = \beta) \right) h(H = \beta | N = j)\, d\beta. \tag{3.22}$$

Proof. Equation (3.20) is trivial; when there is no eigenvalue inside the unit circle, the algorithm will converge to an eigenvalue on or outside the unit circle and terminate.

For $j > 0$ at the first iteration of the algorithm, we compute the closest eigenvalue to the midpoint and repeat the same procedure with the left interval and with the right interval, so the equality

$$X = X_l + X_r + 1$$

and therefore the equality

$$E_j(X) = E(X_l | N = j) + E(X_r | N = j) + 1 \tag{3.23}$$

follow. Clearly the number of iterations on the left and right intervals depend on the modulus of the computed eigenvalue. By the definition of conditional expectations, we deduce

$$E(X_l | N = j) = \int_0^1 E(X_l | N = j, H = \beta) h(H = \beta | N = j)\, d\beta \tag{3.24}$$

and similarly

$$E(X_r | N = j) = \int_0^1 E(X_r | N = j, H = \beta) h(H = \beta | N = j)\, d\beta. \tag{3.25}$$

Now we focus on the procedures applied on the left and right intervals. Let the modulus of the eigenvalue computed at the first iteration be β. There may be up to $j - 1$ eigenvalues inside the circle centered at the midpoint of the left interval $[-1, -\beta]$ and with radius $\frac{1-\beta}{2}$. The expected number of iterations on the left interval is independent of the radius $\frac{1-\beta}{2}$ and the number of eigenvalues lying outside this circle. Therefore given the number of eigenvalues inside this circle, by the definition of conditional expectations, the equality

(3.26)

$$\begin{aligned} E(X_l | N = j, H = \beta) &= \sum_{k=0}^{j-1} E(X_l | N_l = k, N = j, H = \beta) g_l(N_l = k | N = j, H = \beta) \\ &= \sum_{k=0}^{j-1} E(X_l | N_l = k) g_l(N_l = k | N = j, H = \beta) \\ &= \sum_{k=0}^{j-1} E_k(X) g_l(N_l = k | N = j, H = \beta) \end{aligned}$$

is satisfied. A similar argument applies to the right interval to show the analogous equality

$$E(X_r | N = j, H = \beta) = \sum_{k=0}^{j-1} E_k(X) g_l(N_l = k | N = j, H = \beta). \tag{3.27}$$

By substituting (3.26) into (3.24), (3.27) into (3.25), and combining these with (3.23), we deduce the result. ∎

COROLLARY 3.5 (average case for Algorithm 4). *Suppose the eigenvalues of the matrices input to Algorithm 4 are selected uniformly and independently inside the circle of radius μ. The expectation $E_j(X)$ is bounded above by $c\sqrt{j+f}-1$ for all $c \geq \sqrt{12}$ and $f \in [4/c^2, 1/3]$.*

Proof. The proof is by induction. In the base case, when there is no eigenvalue inside the unit circle, the algorithm iterates only once, i.e., $E_0(X) = 1 \leq c\sqrt{f}-1$.

Assume for all $k < j$, that the claim $E_k(X) \leq c\sqrt{k+f}-1$ holds. We need to show the inequality $E_j(X) \leq c\sqrt{j+f}-1$ is satisfied under this assumption. By definition (3.22) in Theorem 3.4 we have

(3.28)
$$F_{j-1}(X) \leq \int_0^1 \left(\sum_{k=0}^{j-1} (c\sqrt{k+f}-1) g_l(N_l = k | N = j, H = \beta) \right) h(H = \beta | N = j)\, d\beta.$$

As we argued before, the uniformity and independence of each of the $j-1$ eigenvalues inside the unit circle but outside the circle of radius $H = \beta$ is preserved. In other words $g_l(N_l|N = j, H = \beta)$ is a binomial density function, and we can explicitly write $g_l(N_l = k|N = j, H = \beta)$, the probability that there are k eigenvalues inside the left circle given that there are $j-1$ eigenvalues contained in the unit circle and outside the circle of radius β, as

$$g_l(N_l = k|N = j, H = \beta) = \binom{j-1}{k} \left(\frac{1-\beta}{4(1+\beta)} \right)^k \left(1 - \frac{1-\beta}{4(1+\beta)} \right)^{j-1-k}.$$

Now the expected value of the binomial distribution above is $(j-1)\frac{1-\beta}{4(1+\beta)}$. From Lemma 3.3, we deduce

$$\begin{aligned} \frac{\sqrt{j+f}}{2} &\geq \frac{\sqrt{j-1+4f}}{2} \\ &\geq \sqrt{\frac{(1-\beta)(j-1)}{4(1+\beta)} + f} \\ &= \sqrt{\sum_{k=0}^{j-1} (k+f) g_l(N_l = k|N = j, H = \beta)} \\ &> \sum_{k=0}^{j-1} \sqrt{k+f}\, g_l(N_l = k|N = j, H = \beta). \end{aligned}$$

Substituting the upper bound $\frac{\sqrt{j+f}}{2}$ for $\sum_{k=0}^{j-1} \sqrt{k+f}\, g_l(N_l = k|N = j, H = \beta)$ in (3.28) yields

$$F_{j-1}(X) \leq \int_0^1 \left(\frac{c\sqrt{j+f}}{2} - 1 \right) h(H = \beta | N = j)\, d\beta = \frac{c\sqrt{j+f}}{2} - 1. \tag{3.29}$$

Now it follows from (3.21) that

$$E_j(X) \leq c\sqrt{j+f} - 1 \tag{3.30}$$

as desired. ∎

Recall that we intend to apply the divide and conquer approach to $\mathcal{A}$ which has size $2n^2 \times 2n^2$. Assume that the conditions of Corollary 3.5 hold for the eigenvalues of $\mathcal{A}$ and the circle of radius D contains all of the eigenvalues. Suppose also that for any shift ν, convergence of the shifted inverse iteration or shift-and-invert Arnoldi method to the closest eigenvalue requires the matrix vector multiplication $(\mathcal{A} - \nu I)^{-1}u$ for various u only a constant number of times. Then the average running time of each trisection step is $O(n^4)$, since finding the closest eigenvalue takes $O(n^3)$ time (which is the cost of solving a Sylvester equation of size $2n$ a constant number of times) and we compute the closest eigenvalue $O(n)$ times at each trisection step on average. Because of the special structure of the Kronecker product matrix $\mathcal{A}$, even if the input matrices have eigenvalues uniformly distributed and mutually independent, the eigenvalues of $\mathcal{A}$ may not have this property. However, the numerical examples in the next section suggest that the number of closest eigenvalue computations as a function of the size of the Kronecker product matrices is still bounded by $O(\sqrt{q})$. According to Theorem 3.2, in the worst case scenario, each trisection step requires $O(n^5)$ operations, which is an improvement over computing all of the eigenvalues of $\mathcal{A}$.

4. Numerical experiments. We first compare the accuracy of the new algorithm with the divide and conquer approach, and Gu's algorithm in [6] on a variety of examples. Second, we discuss why in general we prefer the divide and conquer approach over the adaptive progress approach. In our final example we aim to show the asymptotic running time difference between the new method and Gu's method. All the tests are run using MATLAB 6.5 under Linux on a PC.

4.1. Accuracy of the new algorithm and the old algorithm. We present results comparing the accuracy of the new method using the divide and conquer approach with Gu's method in [6]. In exact arithmetic both the method in [6] and the new method using the divide and conquer approach must return the same interval, since they perform the same verification by means of different but equivalent eigenvalue problems. Our data set consists of pairs (A, B), where A is provided by the software package `EigTool` [13] and B has entries selected independently from the normal distribution with zero mean and variance one. The data set is available on the web.[3] In all of the tests the initial interval is set $[0, \sigma_n([A \quad B])]$ and the trisection step is repeated until an interval $(l, u]$ with $u - l \leq 10^{-4}$ is obtained.

When the second and third columns in Table 4.1 are considered, on most of the examples the methods return the same interval with the exception of the companion, Demmel, Godunov, and gallery5 examples. The common property of these matrices is that they have extremely ill-conditioned eigenvalues. As we discuss in section 5, when the matrix A has an ill-conditioned eigenvalue, the new method is not expected to produce accurate small intervals containing the distance to uncontrollability. One false conclusion that one may draw from Table 4.1 is that Gu's method is always more accurate than the new method. Indeed for the Basor–Morrison, Grcar, or Landau examples with $n = 5$ (for which the eigenvalues are fairly well conditioned) the new method generates more accurate results than Gu's method when one seeks intervals of length around 10^{-6}. In terms of accuracy these two methods have different weaknesses.

[3] http://www.cs.nyu.edu/∼mengi/robust_stability/data_dist_uncont.mat.

TABLE 4.1

For pairs (A, B) with A chosen from `EigTool` *as listed above in the leftmost column and B normally distributed, intervals $(l, u]$ that are supposed to contain the distance to uncontrollability of the system (A, B) are computed with $u - l \leq 10^{-4}$ by both of the methods. The size of the system (n, m) is provided next to the name of the matrix A in the leftmost column. In the fourth column the norm of $\mathcal{A}$ computed by* MATLAB *at the last trisection step is given. In the rightmost column the norm of $\mathcal{A}$ is approximated using* (5.1).

Example	New method	Gu's method	$\|\mathcal{A}\|$	$\approx \|\mathcal{A}\|$
Airy(5,2)	(0.03759,0.03767]	(0.03759,0.03767]	8×10^7	7×10^8
Airy(10,4)	(0.16337,0.16345]	(0.16337,0.16345]	4×10^7	6×10^8
Basor–Morrison(5,2)	(0.68923,0.68929]	(0.68923,0.68929]	2×10^6	2×10^7
Basor–Morrison(10,4)	(0.60974,0.60980]	(0.60974,0.60980]	2×10^7	5×10^8
Chebyshev(5,2)	(0.75026,0.75034]	(0.75026,0.75034]	3×10^7	5×10^8
Chebyshev(10,4)	(0.82703,0.82711]	(0.82703,0.82711]	6×10^{10}	3×10^{12}
Companion(5,2)	(0.42431,0.42438]	(0.42431,0.42438]	2×10^8	4×10^9
Companion(10,4)	(0.46630,0.46637]	(0.46610,0.46616]	5×10^{13}	5×10^{18}
Convection diffusion(5,2)	(0.69829,0.69836]	(0.69829,0.69836]	7×10^5	1×10^7
Convection diffusion(10,4)	(1.48577,1.48586]	(1.48577,1.48586]	9×10^6	1×10^8
Davies(5,2)	(0.23170,0.23176]	(0.23170,0.23176]	2×10^6	2×10^7
Davies(10,4)	(0.70003,0.70012]	(0.70003,0.70012]	1×10^6	1×10^7
Demmel(5,2)	(0.09090,0.09097]	(0.09049,0.09056]	8×10^{49}	`Inf`
Demmel(10,4)	(0.12049,0.12057]	(0.11998,0.12006]	9×10^{80}	`Inf`
Frank(5,2)	(0.45907,0.45916]	(0.45907,0.45916]	1×10^7	7×10^8
Frank(10,4)	(0.67405,0.67414]	(0.67405,0.67414]	3×10^{16}	2×10^{18}
Gallery5(5,2)	(0.17468,0.17474]	(0.02585,0.02592]	1×10^{16}	1×10^{29}
Gauss–Seidel(5,2)	(0.06279,0.06288]	(0.06279,0.06288]	2×10^{20}	`Inf`
Gauss–Seidel(10,4)	(0.05060,0.05067]	(0.05060,0.05067]	1×10^{30}	3×10^{40}
Godunov(7,3)	(1.23802,1.23810]	(1.23764,1.23773]	1×10^{14}	8×10^{31}
Grcar(5,2)	(0.49571,0.49579]	(0.49571,0.49579]	2×10^5	3×10^6
Grcar(10,4)	(0.44178,0.44185]	(0.44178,0.44185]	4×10^7	6×10^8
Hatano(5,2)	(0.39570,0.39578]	(0.39570,0.39578]	4×10^6	2×10^7
Hatano(10,4)	(0.23297,0.23304]	(0.23297,0.23304]	4×10^8	1×10^{10}
Kahan(5,2)	(0.18594,0.18601]	(0.18594,0.18601]	8×10^8	2×10^{10}
Kahan(10,4)	(0.05587,0.05594]	(0.05587,0.05594]	8×10^{13}	7×10^{14}
Landau(5,2)	(0.41766,0.41773]	(0.41766,0.41773]	1×10^5	1×10^6
Landau(10,4)	(0.28166,0.28174]	(0.28166,0.28174]	1×10^7	3×10^8
Markov chain(6,2)	(0.04348,0.04358]	(0.04348,0.04358]	3×10^7	5×10^8
Markov chain(10,4)	(0.07684,0.07693]	(0.07684,0.07693]	8×10^8	6×10^{10}
Orr–Sommerfield(5,2)	(0.04789,0.04796]	(0.04789,0.04796]	1×10^9	8×10^9
Orr–Sommerfield(10,4)	(0.07836,0.07843]	(0.07836,0.07843]	2×10^{10}	2×10^{12}
Skew–Laplacian(8,3)	(0.01001,0.01011]	(0.01001,0.01011]	3×10^{10}	4×10^{13}
Supg(4,2)	(0.06546,0.06554]	(0.06546,0.06554]	7×10^8	4×10^9
Supg(9,4)	(0.03627,0.03634]	(0.03627,0.03634]	2×10^{13}	1×10^{14}
Transient(5,2)	(0.11027,0.11036]	(0.11027,0.11036]	3×10^7	4×10^8
Transient(10,4)	(0.13724,0.13731]	(0.13724,0.13731]	6×10^8	6×10^9
Twisted(5,2)	(0.14929,0.14936]	(0.14929,0.14936]	2×10^7	1×10^8
Twisted(10,4)	(0.77178,0.77185]	(0.77178,0.77185]	1×10^7	2×10^8

TABLE 4.2
A comparison of the real eigenvalue extraction techniques when the matrix A has eigenvalues squeezed in a small real interval.

Method	d	Computed interval	No. of calls to eigs
Adaptive progress	10^{-1}	(0.00702,0.00711]	19
Adaptive progress	10^{-2}	(0.00474,0.00483]	35
Adaptive progress	10^{-3}	(0.00476,0.00484]	81
Divide and conquer	-	(0.00476,0.00484]	13
Gu's algorithm	-	(0.00476,0.00484]	-

TABLE 4.3
A comparison of the real eigenvalue extraction techniques for an uncontrollable pair.

Method	d	Computed interval	No. of calls to eigs
Adaptive progress	1	(0.13538,0.13546]	17
Adaptive progress	0.5	(0.00025,0.00033]	19
Adaptive progress	0.1	(0.00000,0.00008]	50
Divide and conquer	-	(0.00000,0.00008]	12
Gu's algorithm	-	(0.00000,0.00008]	-

4.2. Comparison of the real eigenvalue extraction techniques. When the Kronecker product matrix $\mathcal{A}$ has too many eigenvalues close to the real axis, the adaptive progress method is not the ideal real eigenvalue extraction technique. A remedy for the efficiency problems in this case is choosing d large, which may cause accuracy problems. Suppose we choose $A = Q\,\mathrm{diag}(v)\,Q^*$, where

$$v = [-1 \;\; -0.99 \;\; -0.98 \;\; -0.97 \;\; -0.96]$$

and Q is a unitary matrix whose columns form an orthonormal basis for the column space of a normally distributed matrix. The matrix B is chosen from a normal distribution. The eigenvalue pattern of A is also reflected in $\mathcal{A}$, as it also has eigenvalues tightly squeezed around -1. Among the values listed in Table 4.2, $d = 10^{-3}$ is the one for which the adaptive progress approach returns the correct interval. But the average number of calls to `eigs` for $d = 10^{-3}$ by the adaptive progress approach is more than six times the number of calls made by the divide and conquer approach.

In a second example we choose the pair

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0.95 & 1 \\ 0 & 0 & 0.9 \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ 0.1 \\ 0 \end{pmatrix},$$

which is uncontrollable since $\mathrm{rank}([A - 0.9I \;\; B]) = 2$. In Table 4.3 we see that the adaptive progress approach with $d = 0.1$ yields the correct interval. Nevertheless the number of calls to `eigs` is once again excessive compared to the number of calls by the divide and conquer approach.

These examples illustrate that there may not exist a d value such that the adaptive progress returns an accurate result with fewer calls to `eigs` than the number of calls required by the divide and conquer approach.

4.3. Running times of the new algorithm on large matrices. To observe the running time differences between Gu's method and the new method with the divide and conquer approach, we run the algorithms on pairs (A, B) of various size, where A is a Kahan matrix available through `EigTool` and B is a normally distributed matrix.

TABLE 4.4

Running times of Gu's method/new method in seconds and the average number of calls to eigs by the new method for Kahan-random matrix pairs of various size.

Size (n,m)	t_{cpu} (Gu's method)	t_{cpu} (new method)	No. of calls to eigs
(10,6)	47	171	34
(20,12)	3207	881	63
(30,18)	46875	3003	78
(40,24)	263370	7891	92

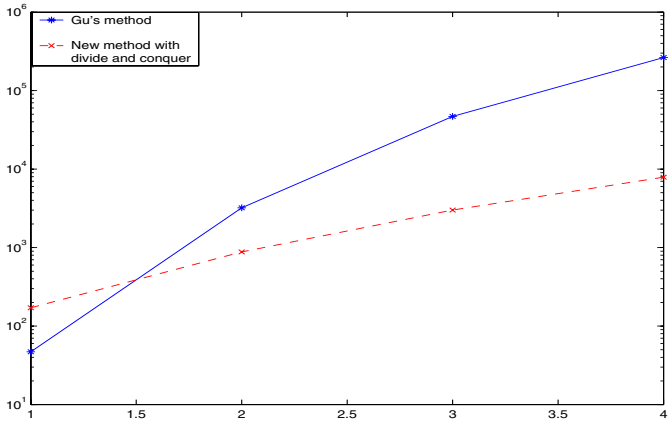

FIG. 4.1. *Running times of the methods on Kahan-random matrix pairs are displayed as functions of the size of the matrix in logarithmic scale.*

We normalized the pairs (by dividing them by $\sigma_n([A\ B])$) so that the same number of trisection steps are required. For $(n, m) = (40, 24)$ we didn't run Gu's method, since it takes an excessive amount of time. Instead we extrapolated its running time. For all other sizes both methods return the same interval of length approximately 10^{-4}. In Table 4.4 the running times of both the algorithms and the average number of calls to eigs made by the divide and conquer approach are provided for various sizes. For small pairs Gu's method is faster. However, for matrices of size 20 and larger the new method is more efficient and the difference in the running times increases drastically as a function of n. In the third column the average number of calls to `eigs` is shown and apparently varies linearly with n. Figure 4.1 displays plots of the running times as functions of n using a log scale. The asymptotic difference in the running times agrees with the plots.

5. Concluding remarks. Based on the results in the previous section, among all the methods discussed, the most reliable and efficient appears to be the new verification scheme with the divide and conquer approach to extract the real eigenvalues. The divide and conquer approach requires only an upper bound on the norm of $\mathcal{A}$. In practice this parameter may be set arbitrarily large and the efficiency of the algorithm is affected insignificantly. Alternatively the upper bound on $\|\mathcal{A}\|$ in [11], or when A has simple eigenvalues, the formula (5.1) discussed below can be used.

Improvements to the divide and conquer approach still seem possible. As the upper and lower bound become closer, the Kronecker product matrices $\mathcal{A}$ in two successive iterations differ only slightly. Therefore it is desirable to benefit from the eigenvalues computed in the previous iteration in the selection of the shifts. We

address further details of the new algorithm below.

5.1. Sylvester equation solvers. The Sylvester equations needed to perform the multiplication $(\mathcal{A}-\nu I)^{-1}u$ are not sparse in general. We solve them by first reducing the coefficient matrices on the left-hand side of (3.17) to upper quasi-triangular forms (block upper triangular matrices with 1×1 and 2×2 blocks on the diagonal). Then the algorithm of Bartels and Stewart can be applied [2]. In our implementation we used the LAPACK routine `dtrsyl` [1], which is similar to the method of Bartels and Stewart, but rather than computing the solution column by column it generates the solution row by row, bottom to top. A more efficient alternative may be the recursive algorithm of Jonsson and Kågström [7].

5.2. Difficulties in computing to a high precision. Gu's method in [6] suffers from the fact that the matrix Q_{12} in (2.4) becomes highly ill-conditioned as $\delta \to 0$ and is not invertible at the limit. This is an issue if the input pair is uncontrollable or nearly uncontrollable.

For the new method instability is caused by small η. The accuracy of the algorithm depends on the ability to extract the real eigenvalues of $\mathcal{A}$ successfully. (The imaginary eigenvalues of $H(\alpha)$ can be obtained reliably by using a Hamiltonian eigenvalue solver.) A computed eigenvalue of $\mathcal{A}$ differs from the exact one by a quantity with modulus on the order of $\|\mathcal{A}\|\epsilon_{mach}/|w^*z|$, where w and z are the corresponding unit left and right eigenvectors, respectively. In general the more dominant factor in the formation of this numerical error is the norm $\|\mathcal{A}\|$ rather than the absolute condition number of the eigenvalue (appearing in the denominator), since the inverted matrix in the definition of $\mathcal{A}$ is the inverse of a matrix that is nearly singular for small η, and therefore the norm of $\mathcal{A}$ is big. There is another numerical trouble caused by big $\|\mathcal{A}\|$. We cannot expect to solve the linear system $(\mathcal{A}-\nu I)x = u$ accurately for $\mathcal{A}$ with large norm. This obviously has an effect on the convergence of shifted inverse iteration and shift-and-invert preconditioned Arnoldi especially considering the fact that the shift ν is not close to an eigenvalue in general. (Because of this, computing the eigenvalues of $\mathcal{A}$ using the QR algorithm may be superior to computing them using shifted inverse iteration or shift-and-invert preconditioned Arnoldi, as indeed we observed in practice.) In our experience `eigs` has convergence problems typically when the norm of $\mathcal{A}$ reaches the order of 10^{10}. Smaller η contributes to the increase in the norm of $\mathcal{A}$; however, it is not the only factor. Indeed for certain pairs (A, B) the norm $\|\mathcal{A}\|$ is large even when η is not small. Under the assumption that A is diagonalizable, an upper bound on $\|\mathcal{A}\|$ is derived in [11]. Specifically when A has simple eigenvalues, the upper bound on $\|\mathcal{A}\|$ in [11] simplifies to

$$2\|A\| + \frac{(2\|BB^*/\delta - \delta I\| + \delta)^2}{\eta \inf_{\det(A-\lambda I)=0} |y_\lambda^* x_\lambda|^2} \tag{5.1}$$

with x_λ and y_λ denoting the unit right and unit left eigenvectors, respectively, corresponding to λ. Notice that the upper bound given by (5.1) can be efficiently computed in $O(n^3)$ time, and therefore in an implementation it can be used to estimate the length of the smallest interval containing the distance to uncontrollability that can possibly be computed accurately. Surprisingly the norm of $\mathcal{A}$ heavily depends on the worst conditioned eigenvalue of A, but it has little to do with the norm of A. For instance, when A is normal and $\|B\|$ is not very large, we expect that $\|\mathcal{A}\|$ exceeds 10^{10} only when η is smaller than 10^{-10} unless the pair (A, B) is nearly uncontrollable. This in turn means we can reliably compute an interval of length 10^{-10} containing

the distance to uncontrollability. On the other hand when A is far from being normal or the pair (A, B) is close to being uncontrollable and a small interval is required, the new method performs poorly. The accuracy of the intervals generated on various examples in Table 4.1 in the second column is also justified by (5.1). All of the examples for which the method performs poorly are highly nonnormal. In the fourth column in Table 4.1 the norms of $\mathcal{A}$ computed by calling MATLAB's `norm` at the last trisection step (approximately when η is the difference between the upper and lower bounds of the interval in the second column and δ is the upper bound of the interval) are listed. In the rightmost column the upper bounds on the norm of $\mathcal{A}$ using (5.1) are provided. For most of the pairs in Table 4.1 the upper bound on $\|\mathcal{A}\|$ in the rightmost column is tight.

5.3. Alternative eigenvalue problem. To see whether there exists an α such that $H(\alpha)$ and $H(\alpha + \eta)$ share an eigenvalue, we extract the real eigenvalues of $\mathcal{A}$. Alternatively we can solve the generalized eigenvalue problem

(5.2)
$$P - \lambda M = \begin{pmatrix} -A_1^* - A_2^T & \delta I & \delta I & 0 \\ B_2^T & -A_1^* + \bar{A}_2 & 0 & \delta I \\ B_1 & 0 & A_1 - A_2^T & \delta I \\ 0 & B_1 & B_2^T & A_1 + \bar{A}_2 \end{pmatrix} - \lambda \begin{pmatrix} -I & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I \end{pmatrix}.$$

The real eigenvalue extraction techniques are applicable to this problem as well, since the scalar λ is an eigenvalue of the pencil above if and only if $\frac{1}{\lambda - \nu}$ is an eigenvalue of the matrix $(P - \nu M)^{-1} M$. The multiplication $x = (P - \nu M)^{-1} M y$ can be performed efficiently by solving the linear system $(P - \nu M)x = My$. When we write this linear system in matrix form, we obtain the Sylvester equation (3.3) but with α replaced by ν and with the matrix

$$\begin{pmatrix} -Y_{11} & 0 \\ 0 & Y_{22} \end{pmatrix}$$

replacing 0 on the right-hand side, where $y = [\mathbf{vec}(Y_{11})\ y_{12}\ y_{21}\ \mathbf{vec}(Y_{22})]^T$ with equal sized block components. Notice that the fact that the eigenvalue problem (5.2) is of double size compared to the eigenvalue problem $\mathcal{A}x = \lambda x$ is not an efficiency concern. We still solve Sylvester equations of the same size. The real issue is that these two eigenvalue problems have different conditioning. Theoretically either of them can be better conditioned than the other in certain situations. In practice we retrieved more accurate results with the eigenvalue problem $\mathcal{A}x = \lambda x$ most of the time, even though there are also examples on which the algorithm using (5.2) yields more accurate results.

6. Software. By combining the new verification scheme and BFGS, it is possible to come up with a more efficient and accurate algorithm. A local minimum of the function $\sigma_n([A - \lambda I \quad B])$ can be found in a cheap manner by means of the BFGS optimization algorithm. Notice that the cost of this local optimization step is $O(1)$, since we are searching over two unknowns, namely, the real and the imaginary parts of λ. Using the new verification scheme we can check whether the local minimum is indeed a global minimum as described in [3, Algorithm 5.3]. If the local minimum is not a global minimum, the new verification scheme also provides us with a point λ', where the value of the function $\sigma_n([A - \lambda I \quad B])$ is less than the local minimum.

Therefore we can repeat the application of BFGS followed by the new scheme until we verify that the local minimum is a global minimum.

An efficient implementation of the new method is freely available.[4] In this implementation, by setting an input parameter appropriately, one can either run the trisection method or the hybrid method just described. Typically, the new scheme is faster than the previous implementation of the trisection method of [3] for matrices of size larger than 20.

Acknowledgements. Many thanks to A. Yılmaz for carefully reading the average case analysis for the divide and conquer approach. We are also grateful to two anonymous referees for their invaluable suggestions.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorenson, *LAPACK Users' Guide*, 3rd ed., SIAM, Philadelphia, 1999.

[2] R. H. Bartels and G. W. Stewart, *Solution of the equation $AX + XB = C$*, Comm. ACM, 15 (1972), pp. 820–826.

[3] J. V. Burke, A. S. Lewis, and M. L. Overton, *Pseudospectral components and the distance to uncontrollability*, SIAM J. Matrix Anal. Appl., 26 (2004), pp. 350–361.

[4] R. Eising, *The distance between a system and the set of uncontrollable systems*, in Mathematical theory of networks and systems, Lecture Notes in Control and Inform. Sci., 58, Springer, London, 1984, pp. 303–314.

[5] R. Eising, *Between controllable and uncontrollable*, System Control Lett., 4 (1984), pp. 263–264.

[6] M. Gu, *New methods for estimating the distance to uncontrollability*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 989–1003.

[7] I. Jonsson and B. Kågström, *Recursive blocked algorithm for solving triangular systems, Part* I: *One-sided and coupled Sylvester-type matrix equations*, ACM Trans. Math. Software, 28 (2002), pp. 392–415.

[8] R. E. Kalman, *Mathematical description of linear dynamical systems*, J. SIAM Control Ser. A, 1 (1963), pp. 152–192.

[9] R. B. Lehoucq, K. Maschoff, D. Sorensen, and C. Yang, *ARPACK Software Package*, http://www.caam.rice.edu/software/ARPACK/, 1996.

[10] R. B. Lehoucq, D. Sorensen, and C. Yang, *ARPACK Users' Guide*, SIAM, Philadelphia, 1998.

[11] E. Mengi, *Measures for Robust Stability and Controllability*, Ph.D. thesis, Courant Institute of Mathematical Sciences, New York, NY, 2006.

[12] C. C. Paige, *Properties of numerical algorithms relating to computing controllability*, IEEE Trans. Automat. Control, 26 (1981), pp. 130–138.

[13] T. G. Wright, *EigTool: A graphical tool for nonsymmetric eigenproblems*, Oxford University Computing Laboratory, Oxford, UK, http://www.comlab.ox.ac.uk/pseudospectra/eigtool/, 2002.

[4] http://www.cs.nyu.edu/∼mengi/robust_stability/dist_uncont.html.